DOCUMENT RESUME

ED 210 158 SE 035 834

AUTHOR Brooks, Sarah
TITLE Use of the Computer in Solving Mathematics Problems.
PUB DATE 81
NOTE 10p.; Paper presented at the Annual Meeting of the American Mathematical Association of Two Year Colleges (1981).

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computers; Computer Science Education; Educational Technology; *Logic; Mathematical Applications; *Mathematics Education; *Mathematics Instruction; *Problem Solving; *Programing
IDENTIFIERS Computer Logic; Heuristic Methods; *Heuristics; Structured Programing

ABSTRACT

Computer programing and problem-solving steps in mathematics are viewed to have related concepts. Some heuristics are compared with some suggestions for structured programing. The one fundamental difference between problem solving in general and when using the computer is seen as the computer solution must be finite. The computer is viewed as a useful tool in solving those problems where the method of solution is a finite step-by-step procedure. Many examples of areas of study where the computer would be a useful tool in investigation are identified. It is suggested that teachers who include computer programing in their problem-solving classes may find pupils learn programing and problem solving more quickly and easily than they would learn either separately. (MP)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED210158

Use of the Computer in Solving Mathematics Problems

by
Sarah Brooks

Computer programming and solving mathematics problems are synergetic actions. Each supplements and strengthens the other in many ways. For example, recently there has been much emphasis on improved programming techniques: top down design, structured programming, stepwise refinement[4]. Analysis of these techniques, however, reveals that they are concepts of problem solving applied to computer programming. For example, compare the problem solving steps outlined in the 1980 NCTM yearbook with HIPO (Hierarchy of Input-Process-Output), one version of top down programming.

The hierarchy of problem solving steps[1]	HIPO
1. Understand the problem. What is the unknown? What are the data? What is the condition?	Input
2. Devise a plan. Find a connection between the data and the unknown. Obtain a plan of the solution.	Process
3. Carry out your plan.	Output
4. Examine the solution obtained.	Top down testing

Let's compare some heuristics, or problem solving strategies, with some suggestions for structured programming.

SE 035 834

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Sarah Brooks

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Heuristics[2]

1. Draw a diagram.
2. Examine special cases.
3. Plan solutions hierarchially.
4. Be able to explain, at any point in a solution, what you are doing and why; what you will do with the results of this operation.
5. Consider a variety of similar problems.
   a. Replace conditions with simple ones. Relax a condition and then try to reimpose it. Examine analogous problems with less complexity (fewer variables). Explore the role of just one variable or condition, leaving the rest fixed.
   b. Reformulate the problem by:
      1) A change of perspective or notation.
      2) Arguing by contradiction or contrapositive.
      3) Assuming a solution and determining the properties it must have.
   c. Recombine elements of the problem in different ways. Introduce auxiliary variables.

Programming

1. Draw a structure chart.
2. Program to cover all special cases.
3. Use top down programming.
4. Document.
5. Do not reinvent the wheel. Incorporate sections of previously coded programs if they are applicable. KISS. (Keep it simple, stupid.)

d. Exploit any problem with similar form, data, conclusions; try to exploit both the method and the results.

6. Choose subgoals and try to attain them. Decompose the problem and work on it case by case.

6. Use stepwise refinement.

7. Verify the solution. Does it:

   a. Use all the data?

   b. Conform to reasonable estimates?

   c. Withstand tests of symmetry, dimension analysis, scaling?

   Can it be:

   a. Obtained differently?

   b. Substantiated by special cases?

   c. Reduced to known results?

7. Test as many branches of the program as is possible in the given time.

There is one fundamental difference between problem solving in general and problem solving using the computer. The computer solution must be finite. It must be an algorithm: a solution to the problem in a finite sequence of well-defined steps. Therefore the computer is a useful tool in solving those mathematics problems where the method of solution is a finite step by step procedure. Some examples are:

1. Trial and error
2. Patterns
3. Solving a simpler problem
4. Working backwards
5. Simulation

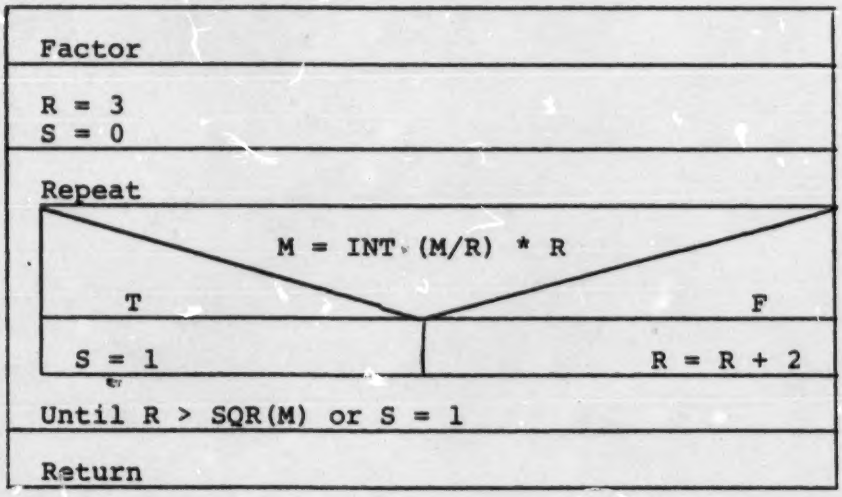
In each of these, the broad outline of the solution must

be created by the human; the computer can only perform the laborious calculations and comparisons necessary to produce a specific solution. Problems may be solved by a combination of human brain and computer brawn.

Suppose you were asked to find the smallest prime number greater than 840. The computer could test every integer greater than 840, and print the first integer which was not composite. But you would have to write a program instructing the computer to test each odd number n greater than 840 by seeing if n could be divided by every odd integer r from 3 to the square root of n, and to print the first integer n which could not be divided by any r.

The structure chart of the computer solution would look like this:

```
Job  PRIME 840
Vars
     N the lower bound of the prime number
     M the prime number
     R the divisors of M
     S A variable switch.  When S = 0, M is prime.
N = 840
M = N + 1
Repeat
  Call Factor
  M = M + 1
Until S = 0
PRINT 'M IS  '; M
END
```



This program could easily be modified to work for any number--it need not be restricted to 840.

Another set of problems which can be solved by trial and error are the cryptarithms, or alphametics, such as $(HE)^2 = SHE$, or $\begin{array}{r} WRONG \\ +WRONG \\ \hline RIGHT \end{array}$, where each letter represents a unique digit. There are two operations which must be part of any computer program solving an alphametic:

1. break the product or sum of any intermediate operation into digits.
2. test each integer replacing a letter to make sure there are no repetitions.

Once one has written one such program, s/he can adapt it to solve any alphametic. These are a lot of fun to do. I should warn you--they are addictive. I solve every alphametic in the MATYC Journal and the Two Year College Mathematics Teachers Journal. The latest one was the division $\begin{array}{r} YOU) \overline{AND} \\ ASK \\ \hline N \end{array}$ (I .

While experiencing the joys of problem solving, the student working with alphametics learns such concepts as digit, place value, and literal expansion in the decimal number system. To see an alphametic solving computer program, read the article "Computer Solution of Alphametics"[5], or write to me and ask for a copy of the program.

The computer is useful also in performing arduous computations and printing results so that one may search for patterns in problems. For example, suppose you wanted to find the sum of the first n odd numbers. You could program the computer to compute the sum s of n odd numbers, $1 \leq n \leq 25$, and print s and n, so that you could try to discover a pattern. I think anyone seeing 1 1; 2 4; 3 9;...; 25 625 would guess that the sum of n odd numbers is n^2.

A structure chart for such a program is:

```
Job  SUMODDS
Vars

   N the number of positive odd integers
   S the sum of the N integers
   I the centrol variable of the FOR...NEXT loop
N = 25
S = 0
FOR I = 1 TO N
    S = S + 2 * I - 1
    PRINT I, S
NEXT I
END
```

Given the problem: Find y such that $*y^2 + \frac{1}{y^2} - 18(y + \frac{1}{y}) - 17 = 9$, one might replace * by its equivalent $(y + \frac{1}{y})^2 - 18(y + \frac{1}{y}) - 19 = 0$, and let $x = y + \frac{1}{y}$. Then by executing a program which solves quadratic equations, the computer could find the roots 19 and -1 for $x^2 - 18x - 19 = 0$, and $\frac{19 \pm \sqrt{357}}{2}$ for $y + \frac{1}{y} = 19$. Thus the emphasis in shifted from the solution of quadratic equations to searching for patterns in problems which can be expressed as quadratic equations.

Programming a computer to play games is an entertaining method of practicing the looking backward technique. Let us illustrate with 'Force Out', a two-player game. Suppose the computer generates a pseudo-random number, N. Then each player may, in turn, subtract any one-digit number from N, until one (the loser) leaves a remainder of zero. If you are to be the first player (A) each time, (the computer is the second player (B)) how can you develop a winning strategy?

Analysis of 'Force Out' strategy:

Last step: B must take one.

Next to last step: A leaves one.

Next to next to last step: B leaves a number from 2 to 10.

Next to next to next to last step: A leaves 11.

How can you ensure that A leaves a remainder of 11? Play the game with the computer for a while to observe the pattern and develop the strategy.

One of the most important applications of computer programs today is simulation. Here is an example:

Each package of Peppy Pops cereal contains one dish. How many boxes of Peppy Pops would you buy in order to collect one

five piece setting of dishes?

Let the computer generate a pseudo-random number from 1 to 5, inclusive. Then count the number of repetitions needed in order to generate all five numbers. Count this as one trial. Calculate the average number of packages purchased over many trials.

This type of problem solving is being applied to many real life problems in business, science, and industry.

As the price of computers decreases, and the ability to program problem solving applications increases, there will be more and more pressure on us as mathematics teachers to incorporate the computer into the classroom. As a matter of fact, by 1985 we may find many, if not most, of our students possess personal (also called micro) computers.

Indeed, if you start now to include computer programming in your problem solving classes, using the techniques of structured programming, you may find the students learn programming and problem solving more quickly and easily than they would learn either separately. There's a whole new wave of ideas and teaching techniques coming. We in mathematics education must prepare now to meet that challenge or we will be swept aside and discarded as irrelevant.

Bibliography

1) Problem Solving in School Mathematics, 1980 Yearbook, Edited by Stephen Krulik and Robert E. Reys, 1980, The National Council of Teachers of Mathematics, 1906 Association Drive, Reston, Virginia, 22091.

2) Schoenfeld, Alan H., Heuristics in the classroom, article in above.

3) Musser, Gary L., Shaughnessy, J. Michael, Problem-solving strategies in school mathematics, article in above.

4) Bohl, Marilyn, A Guide for Programmers, 1978, Prentice-Hall, New Jersey, 07632.

5) Brooks, Sarah, Two Year College Mathematics Journal, March, 1980, Computer Solution of Alphametics.